CS 403, Assignment 1

Due on 11 October at 11:59 pm

In this assignment we will play with polynomials. We represent a polynomial $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$ in HASKELL as a list of pairs (a_i, i), $0 \leq i \leq n$. Whenever $a_i = 0$ the respective pair will not be present in the list. Note that the list representation of a polynomial is not necessarily sorted in any way. You may take `[]` (the empty list) to be a representation of the identity element for polynomial addition.

We refer henceforth to this representation of polynomials as the *coefficient–index list*. For example the polynomial $2x^3 + 6x^2 + 3x + 4$ may be represented by several coefficient–index lists including `[(2,3),(6,2),(3,1),(4,0)]` and `[(4,0),(3,1),(6,2),(2,3)]` (and for that matter everything in between). The polynomial $2x^3+3x$ may be represented by either `[(3,1),(2,3)]` or `[(2,3),(3,1)]`.

It turns out that a different representation of polynomials is much more convenient for operating on them in HASKELL. We call this implementation the *full coefficient list*. The full coefficient list of a polynomial $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$ is the list `[a0,a1,...,an]` of all the coefficients (including null coefficients) sorted in increasing order of their index.

We start the assignment by providing conversion functions between the two representations of polynomials. For this purpose:

1. Modify the Quicksort implementation presented in class so that it sorts lists of pairs `(a,i)` with respect to the second component `i`, thus obtaining the function `polySort`.

2. Define a function `polyExpand` that adds the missing (null) coefficients to a polynomial. You can assume that the polynomial is already sorted in increasing order of its coefficients. For example

   ```
   polyExpand [(3.0,1),(2.0,3)]
   ```

   should evaluate to `[(0.0,0),(3.0,1),(0.0,2),(2.0,3)]`.

3. *Use a map* to define a function `polyList` that receives the output of `polyExpand` and drops the indices, thus obtaining the full coefficient list representation of the polynomial. For example:

   ```
   polyList [(0.0,0),(3.0,1),(0.0,2),(2.0,3)]
   ```

   should evaluate to `[0.0,3.0,0.0,2.0]`.

4. Put all the functions above together to define the function `polyConvert` that receives the coefficient–index list representation of a polynomial and evaluates to the full coefficient list representation of that polynomial. For example:

```
polyConvert [(2.0,3),(3.0,1)]
```

should evaluate to `[0.0,3.0,0.0,2.0]`.

5. *Use a zip and a filter* to define one more function `polyUnconvert` that receives the full coefficient list representation of a polynomial and returns a coefficient–index list representation of that polynomial. For example:

   ```
   polyUnconvert [0.0,3.0,0.0,2.0]
   ```

   should evaluate to either `[(2.0,3),(3.0,1)]` or `[(3.0,1),(2.0,3)]` at your discretion.

We are now ready to manipulate polynomials. All the function below receive polynomial arguments in their coefficient–index list representation and return polynomial values using the same representation. However, they should work internally using the full coefficient list representation for polynomials. Therefore `polyConvert` should be called at the beginning and `polyUnconvert` at the end as applicable. For example the function `polyAdd` implemented for Question 7 below should start something like this:

```
polyAdd px py = polyUnconvert (polyAdd' (polyConvert px) (polyConvert py))
    where polyAdd' px py = -- your actual function here --
```

6. Define a function `polyEval x p` that evaluates the polynomial `p` at `x`. For example:

   ```
   polyEval 2.0 [(3.0,2), (2.0,1), (1.0,0)]
   ```

   should evaluate to 17.0. *Use a fold and Horner's evaluation scheme*:

   $$a_0 + x \times (a1 + \cdots + x \times (a_{n-1} + x \times a_n) \cdots)$$

7. Define a function `polyAdd p q` that returns the sum of the polynomials `p` and `q`. For example,

   ```
   polyAdd [(3.0,2), (2.0,1), (1.0,0)] [(3.0,1), (2.0,0)]
   ```

   should evaluate to `[(3.0,0),(5.0,1),(3.0,2)]` (or equivalent).

8. Define a function `polyMult p q` that returns the product of p and q. For example:

   ```
   polyMult [(3.0,2), (2.0,1), (1.0,0)] [(3.0,1), (2.0,0)]
   ```

   should evaluate to `[(2.0,0),(7.0,1),(12.0,2),(9.0,3)]` (or equivalent). Use the fact that when $n > 0$, $a_0 + a_1 \times x + a_2 \times x^2 + \cdots + a_n \times x^n$ is expressible as $a_0 + x \times (a_1 \times + a_2 \times x + \cdots + a_n \times x^{n-1})$ and thus provide a recursive definition that *uses polynomial addition (implemented earlier) and a map*.

Important notes

The use of programming techniques (folds, maps, and so on) is mandatory whenever specified in the handout. For your convenience these requirements are emphasized throughout the handout.

Provide suitable type definitions for all your top level functions. You can consider that we are only interested in polynomials with `Float` coefficients and `Int` indices. However attempting to provide general, explicit type definitions for your functions is encouraged and success on the matter will result in bonus marks.

Submission guidelines

Submit a single plain text file that can be loaded in the HASKELL interpreter. It would be nice if you can submit a literate script (`.lhs` extension), but this is not required. Your script will only be loaded in the interpreter and will not go anywhere near the compiler, so do not provide anything having to deal with the HASKELL compiler such as a `main` function or I/O operations.

Submit by email your script as explained above Also provide (as suitable comments) session listings or equivalent that demonstrate and test your functions. Include at the top of your script a comment with the names and emails of all the collaborators.

Recall that assignments can be solved in groups[1] (of maximum three students), and a single solution per group should be submitted. Also recall that a penalty of 10% per day will be applied to late submissions.

[1]It is often easier to learn new concepts by working together in a group—you may learn more if you work with smart people. This being said, the purpose of group assignments is that all people in a group should be involved in creating the solution. You can of course choose to sign an assignment you haven't done, but then you risk not learning the material, and this will come back to haunt you during the mid-term examination.